Nota de Clase de Ayudantía: Triangulaciones
Geometría Computacional

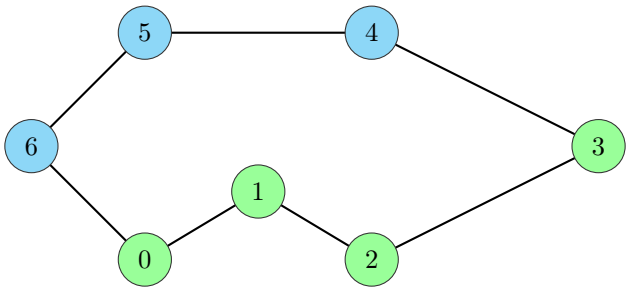
Adriana Ramírez Vigueras
Diana Isabel Ramírez García Ailyn Rebollar Pérez

2 de Abril de 2020

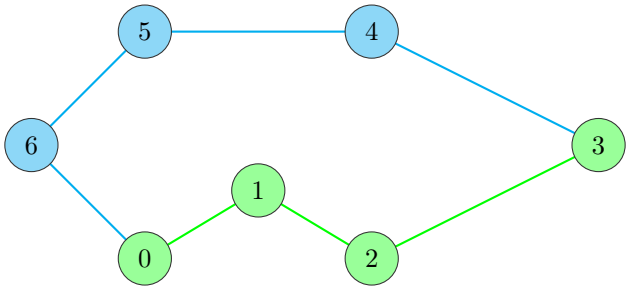
Triangulación de Polígonos Simples

Conceptos Generales

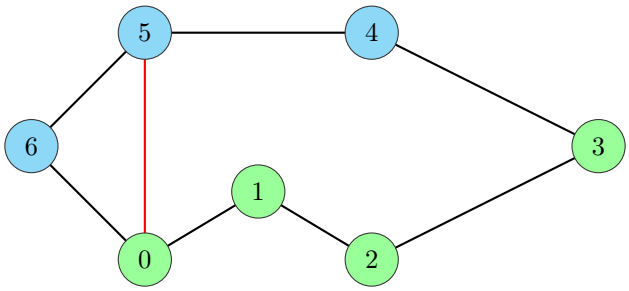
Un *polígono simple* es un polígono donde sus aristas no se intersectan o cruzan entre sí. Un ejemplo de polígono simple es el siguiente:



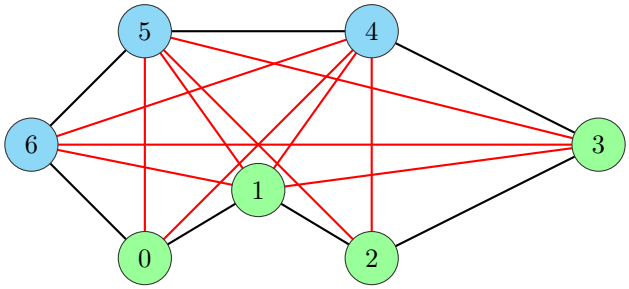
Notemos que los vértices v_0, v_1, v_2 y v_3 están coloreados de verde mientras que los vértices v_4, v_5 y v_6 son azules, esto con el propósito de introducir el concepto de *cadenas poligonales* que son conjuntos de segmentos de recta (en éste caso aristas) consecutivos unidos por vértices. Entonces podemos decir que $\overline{v_0v_1}$, $\overline{v_1v_2}$, $\overline{v_2v_3}$ pertenecen a una cadena poligonal $\mathcal{A}$ que están coloreados de verde y $\overline{v_3v_4}$, $\overline{v_4v_5}$, $\overline{v_5v_6}$ y $\overline{v_6, v_0}$ a otra cadena $\mathcal{B}$ que están coloreados de azul.



Otro concepto que nos interesa para el tema de triangulaciones es el de una *diagonal*, ¿qué es una diagonal de un polígono? Es un segmento de línea que conecta dos vérices del polígono dado y que permanece dentro del interior de éste. Entonces cabe resaltar que una diagonal no es una arista el polígono. Un ejemplo de diagonal sería el segmento d coloreado de rojo que va del vértice v_0 al v_5.



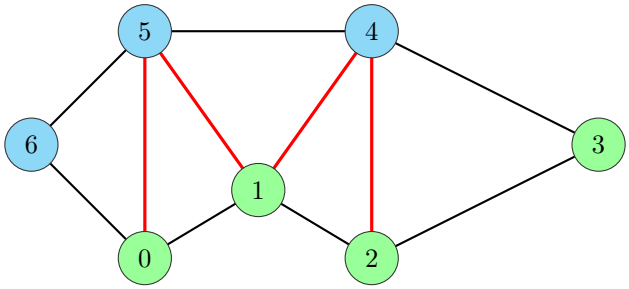
Haciendo todas las diagonales posibles, el polígono quedaría de la siguiente manera.



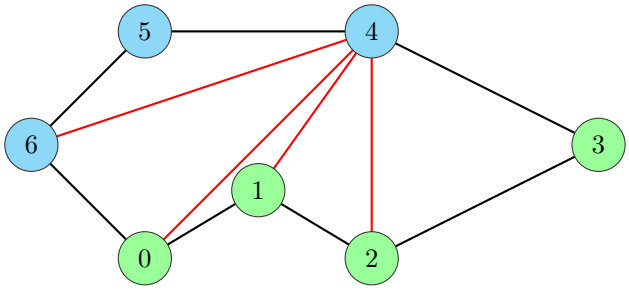
Una vez que ya quedaron claros estos conceptos podemos hablar de triangulaciones.

Defínción de Triangulación de un Polígono

Si fueron observadores al contemplar la figura anterior mostrada, se puede ver como al ir dibujando diagonales el polígono se va descomponiendo en triángulos. Sin embargo, muchas diagonales se intersectan y no permiten que se vean dichos triángulos, entonces, ¿qué pasa si nos fijamos solamente en diagonales que no se intersectan? Un posible resultado sería el siguiente polígono



Lo cual nos permite aterrizar o llegar a la definición de *triangulación*. Una triangulación de un polígono es la descomposición del polígono en triángulos dada por el número máximo de diagonales que puede tener. Las triangulaciones no siempre son únicas, justo como en el polígono que hemos estado viendo, ya que otra posible triangulación podría ser.



Una definición alterna es que una *triangulación* de un conjunto de puntos $\mathcal{S}$ es una subdivisón plana determinada por el máximo número de diagonales que no se intersectan.

Propiedades

En clase se vieron dos resultados importantes que son propiedades para triangulaciones de polígonos de n puntos con $n > 3$ que son:

1. A lo más se pueden tener n-2 triángulos.
2. A lo más se pueden tener n-3 diagonales.

Asimismo, se vio lo que es la *complejidad geométrica* que está dada por $\#V + \#F + \#E$ donde V son los vértices, F son las caras y E aristas es $O(n)$.

Aplicación: Problema de la Galería de Arte

La aplicación más conocida donde se ocupa la triangulación de polígonos simples es en el Problema de la Galería de Arte, en el cual se puede ver el espacio de ésta en forma de polígono simple en el plano y lo que se busca es cubrir toda el área con cámaras de seguridad. Además se busca que el número de cámaras que se utilicen sea mínimo.

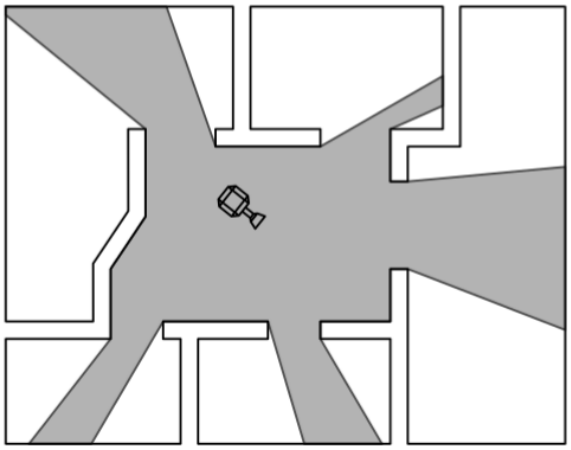


Figure 1: Imagen tomada del libro *Computational Geometry: Algorithms and Applications* de

Algoritmos de Triangulaciones para un Conjunto de Puntos

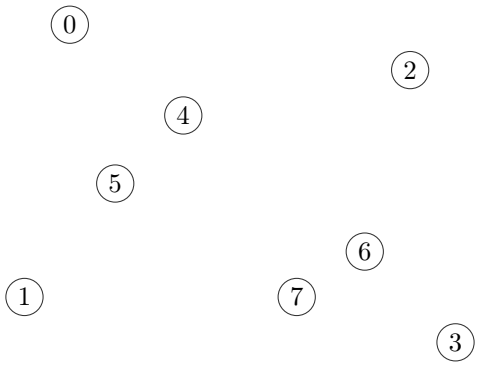
Triangle Splitting

Para el conjunto de puntos en posición general $\mathcal{S}$

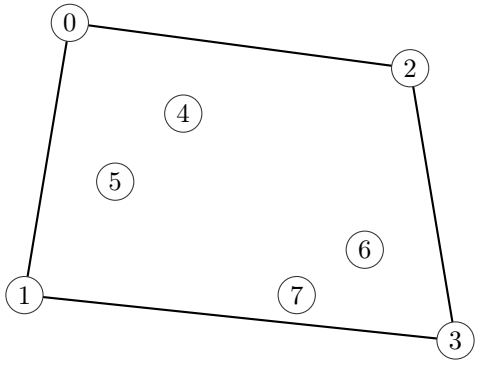
1. Encontrar el cierre convexo de $\mathcal{S}$.
2. Triangulamos a $\mathcal{S}$ como si fuera un polígono convexo.
3. Escoger 1 punto interior p del $\mathcal{CH}$.
4. Dibujar aristas de p a los tres vértices del triángulo en el que está contenido.
5. Repetir desde el paso 3, hasta que ya no haya puntos dentro de algún triángulo.

Un ejemplo de la ejecución de éste algoritmo es:

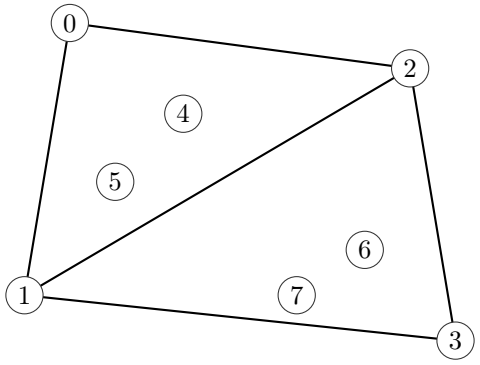
Sea $\mathcal{S}$ el siguiente conjunto de puntos en posición general.



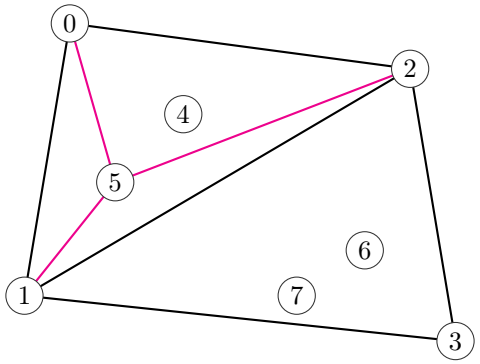
Realizando el paso 1 tenemos:



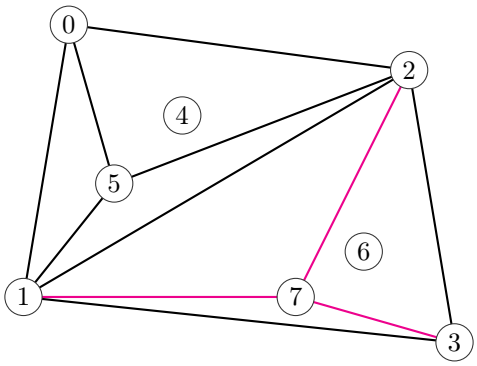
Realizando el paso 2 obtenemos:



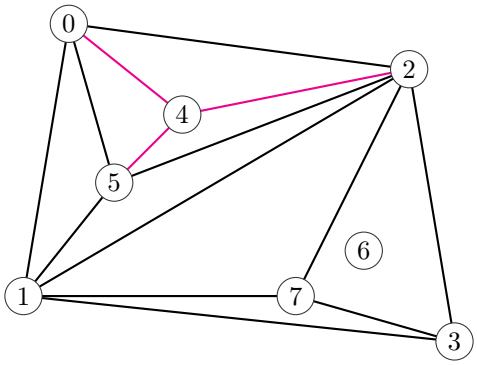
Tomemos el vértice v_5 y realizamos el paso 4.



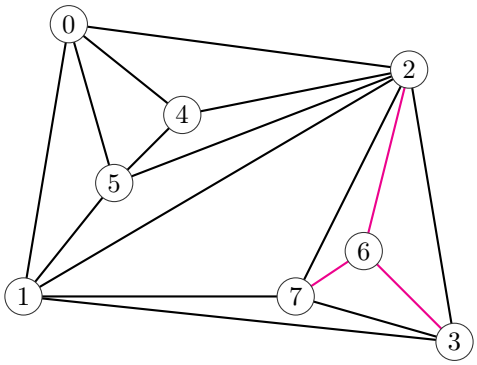
Elegimos al punto v_7 y trazamos las aristas de éste vértice a v_1, v_3 y v_2



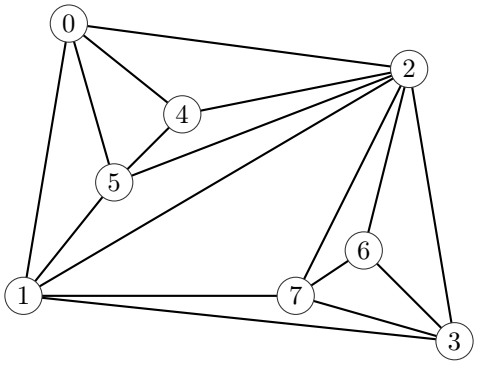
Sucesivamente escogemos al v_4 y trazamos las aristas correspondientes a v_0, v_5 y v_2 que forman al triángulo donde se encuentra.



Finalmente tomamos al punto v_6 y volvemos a repetir el paso 4.



Por lo que ejecutando éste algoritmo obtuvimos ésta triangulación llamada T



Éste algoritmo da pie a diferentes preguntas.

- Al ejecutar nuevamente Triangle Splitting, ¿podemos asegurar que nos regresará la triangulación T? ¿Por qué?
- ¿Cuál es la complejidad de éste algoritmo?

Respuestas:

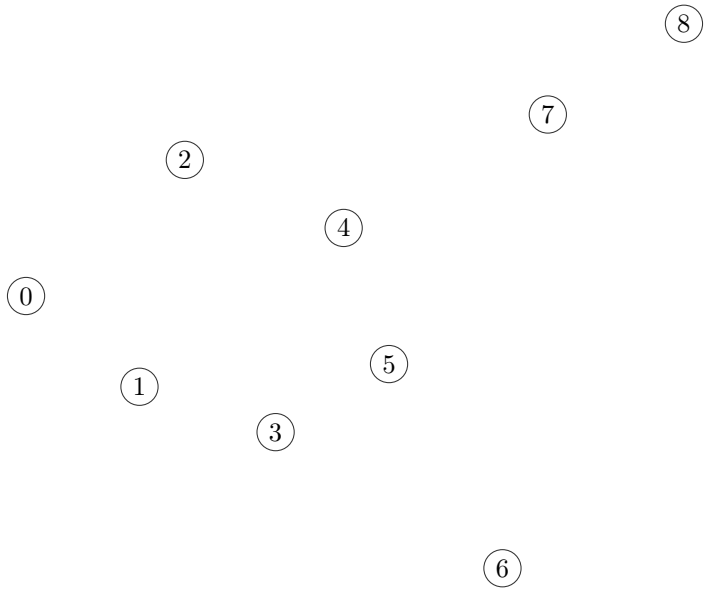
- No podemos garantizarlo, pues siempre vamos tomando un punto al azar para ir trazando nuevos triángulos. O incluso si el algoritmo para triangular el polígono convexo da diferentes triangulaciones, desde ese paso no se garantiza.
- O(n^2) porque el paso 1 nos toma $O(n\log n)$ donde n es el número de puntos, el paso 2 toma $O(n)$, el paso 4 es $O(1)$ pero el paso 5, ¿cuánto toma? Supongamos que m puntos son los que pertenecen al cierre convexo, entonces tenemos que procesar n - m puntos que son los que están en el interior que es una cantidad lineal de puntos, ahora, ¿qué pasa por cada punto? tenemos que localizar el triángulo en el que se encuentra, al inicio el paso 3 nos dejó m-2 triángulos que también es una cantidad lineal, pero cada que trazamos una diagonal siempre vamos encontrando 2 triángulos nuevos que se añaden a la triangulación. Entonces estamos ejecutando un número lineal de pasos que tarda $O(n)$ que es buscar a que triángulo pertenece..

Algoritmo Incremental

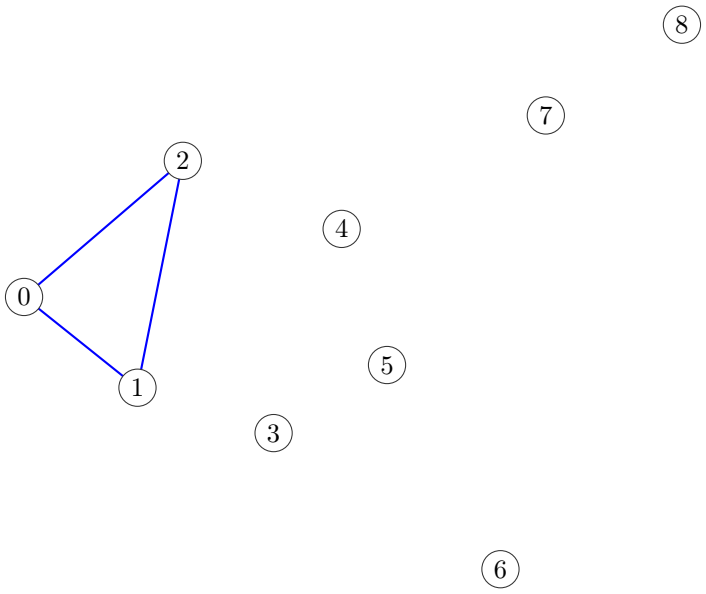
Para el conjunto de puntos de tamaño n en posición general $\mathcal{S}$.

1. Se ordena el conjunto de puntos de acuerdo a la coordenada x.
2. Tomamos los primeros tres puntos y formamos un triángulo.
3. Supongamos que el último punto que formó el triángulo fue p_i, entonces tomemos a p_{i+1} de la lista ordenada de puntos y checamos con $\{p_0, p_1, \ldots, p_i\}$ cuáles son visibles para p_{i+1} y los conectamos a él.
4. Repetir el paso 3 hasta que ya no haya puntos por procesar.

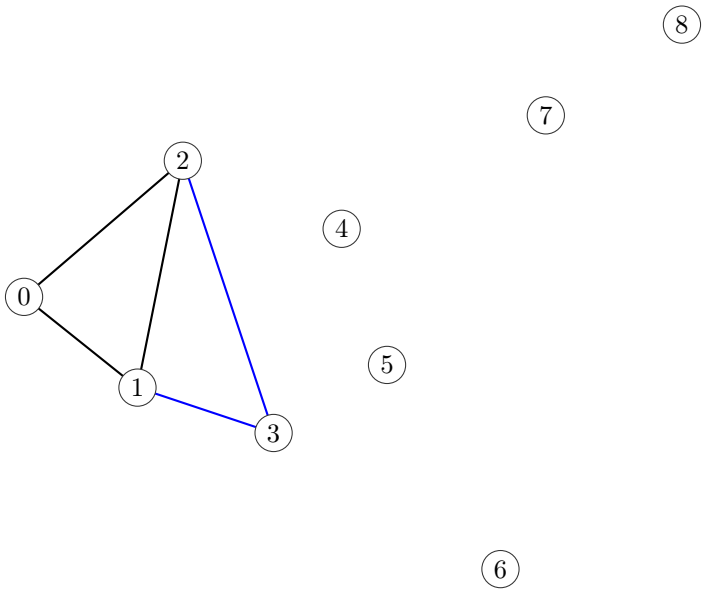
Para ver la ejecución de éste algoritmo tomemos el siguiente ejemplo como el conjunto de puntos $\mathcal{S}$ en posición general y ya ordenados.



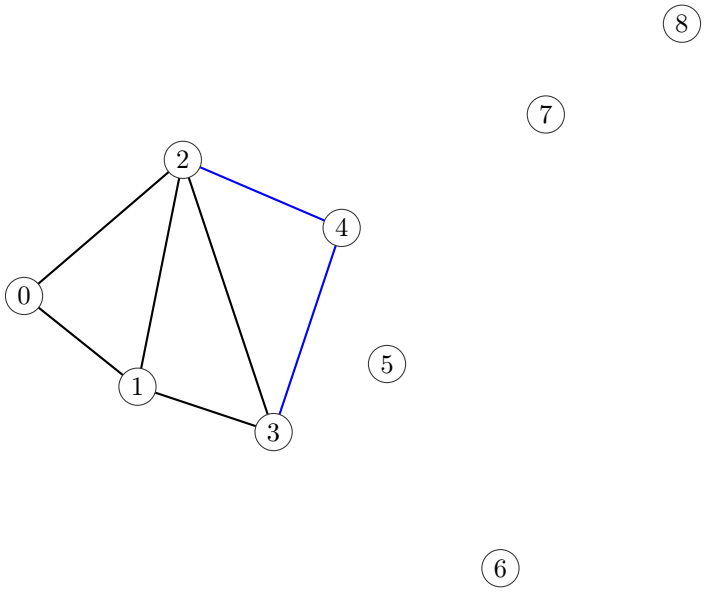
De acuerdo al paso 2, hay que tomar los tres primeros puntos los cuales son p_0, p_1 y p_2 para formar un primer triángulo que sería el dibujado de azul.



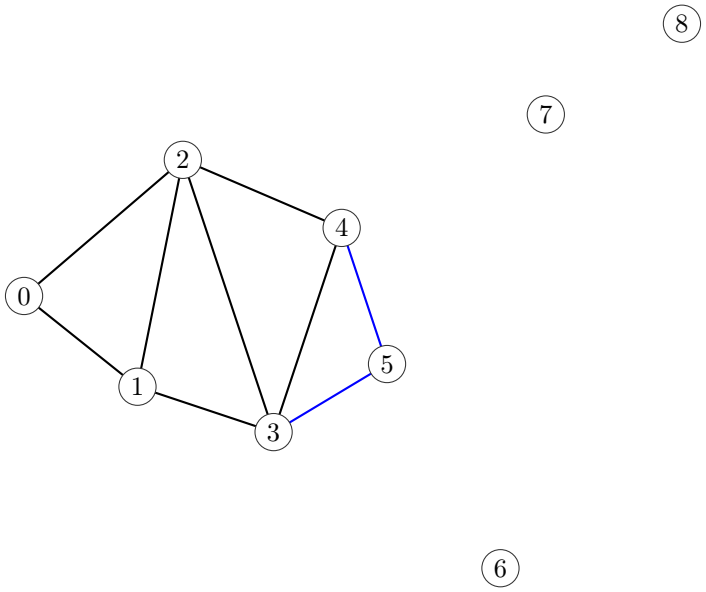
Después, el algoritmo dice que nos tomemos al siguiente punto que es p_3 y veamos a qué puntos tiene "visible" o puede ver para trazar segmentos de recta que los conecten. Realizando esto obtenemos la siguiente figura:



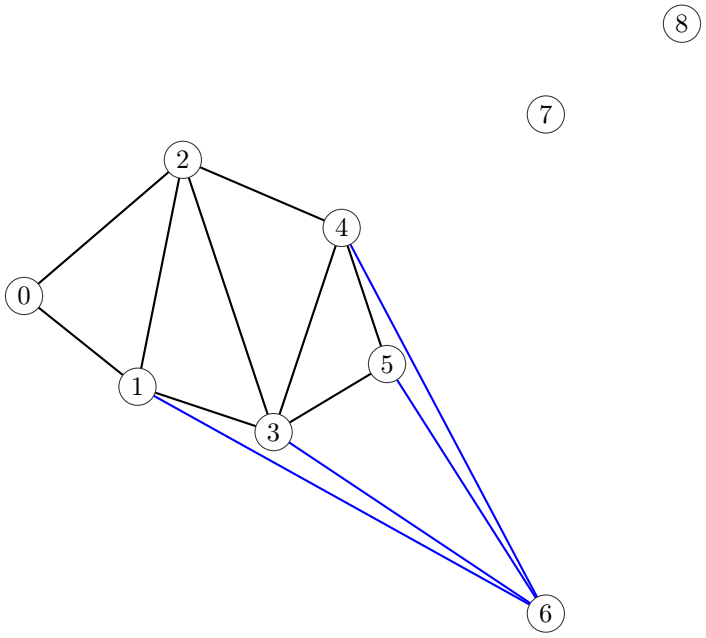
Repitiendo el mismo procedimiento pero para p_4 vemos que es "visible" con p_2 y p_3, así que los conectamos.



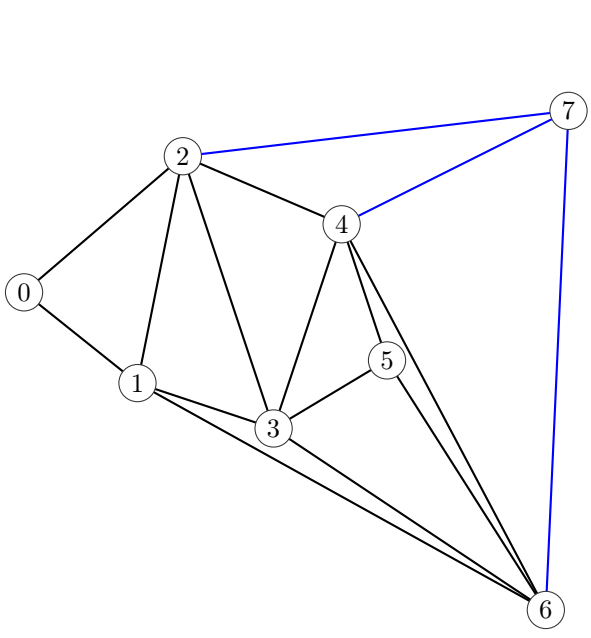
Ahora tomando a p_5, lo conectamos con p_3 y p_4.



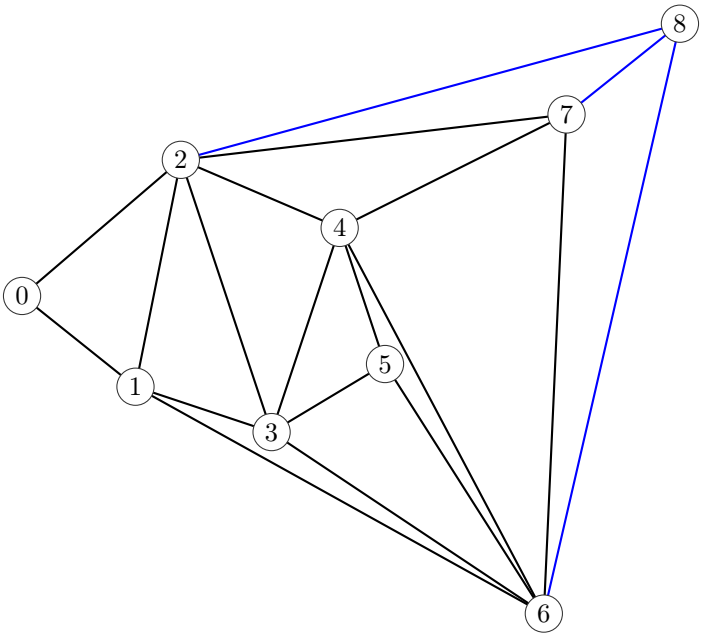
Consideremos a p_6 donde a los puntos que ve son p_4, p_4, p_3 y p_1. Ahora tomando a p_5, lo conectamos con p_3 y p_4.



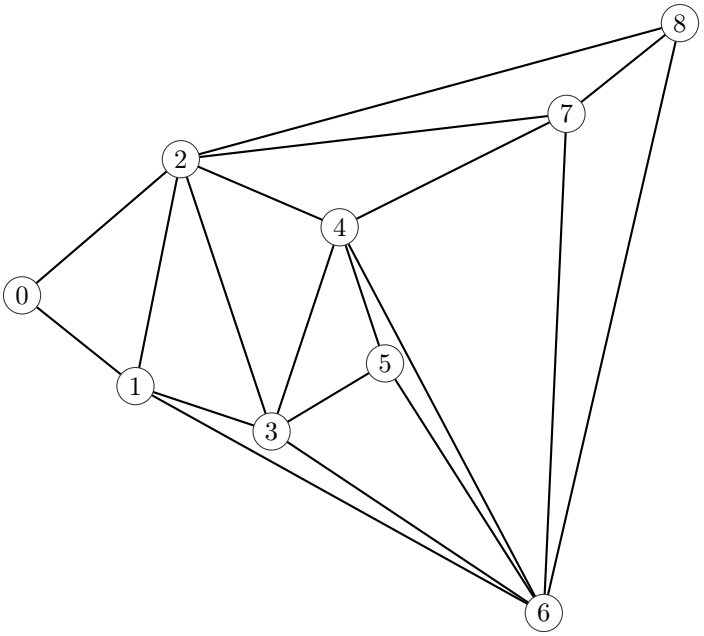
Encontrando los puntos visibles para p_7 y conectándolos queda de la siguiente manera:



Finalmente tomemos a p_8 y lo conectamos con p_2, p_7 y p_6.



Obteniendo así la triangulación T'



Respondamos las mismas preguntas que se hicieron para el algoritmo anterior.

- Al ejecutar de nuevo el algoritmo incremental, ¿podemos asegurar que nos regresará la triangulación T'? ¿Por qué?
- ¿Cuál es la complejidad de éste algoritmo?

Respuestas:

- Al ejecutar de nuevo el algoritmo incremental, sí podemos asegurar que nos regresará T' porque al ordenar otra vez, obtendremos la misma lista de puntos para procesar porque no se agregaron nuevos puntos ni se modificaron.
- La complejidad del algoritmo incremental es de orden $O(n^2)$ ya que el paso de ordenar nos toma $O(n\log n)$ pero el paso 3 se ejecuta $n-3$ veces porque se repite para cada punto en $\mathcal{S}$ además notemos que ese mismo paso toma $O(n)$ debido a que tenemos que revisar a qué puntos puede ver o a cuáles tiene visible para poder lanzar los segmentos de recta que serán las diagonales de triangulación. Entonces, volvemos a ejecutar un número $O(n)$ de pasos que tardan $O(n)$ cada uno.

Referencias

1. Overmars, M. *Computational Geometry. Algorithms and Applications.*3ra Edición. Springer. 2008.
2. Devadoss S. L. *Discrete and Computational Geometry.* Décima Edición. Princeton Univesity Press. 2011